Del A (obligatorisk för alla)

A1. Ringa in rätt svarsalternativ eller skriv svar i ruta om sådan

a) Vad blir resultatet av följande kod?

```
double h = (double)(1/10);
double s = 0;
for (int i= 1; i<=10; i++) {
   s += h;
}
```

1) Kompileringsfel
2) `RuntimeException`
2) `s` får värdet `0.0`
3) `s` får värdet exakt `1.0`
4) `s` får ett värde nära `1.0` men ej exakt

b) Satsen

```
int x = (int)(1. + (9/10 + 9/10)) - 1.;
```

resulterar i

1) Kompileringsfel
2) `x` får värdet `0`
3) `x` får värdet `1`
4) `x` får värdet `2`

c) Vad händer när nedanstående kod körs)

```
int[] a = new int[3];
a[1] = 2;
a[2] = 3;
System.out.println(a[0]*a[1] + a[2]);
```

1) Kompileringsfel
2) `ArrayIndexOutOfBoundsExcepion`
3) `NullPointerException`
4) Utskrift av 0
5) Utskrift av 3
6) Utskrift av 5

d) Vad händer när nedanstående kod körs?

```
ArrayList<Integer> a = new ArrayList<Integer>();
a.add(2);
a.add(3);
System.out.println(a.get(0)*a.get(1) +
                   a.get(2));
```

1) Kompileringsfel
2) `IndexOutOfBoundsExcepion`
3) `NullPointerException`
4) Utskrift av 0
5) Utskrift av 3
6) Utskrift av 5

e) Vad skrivs ut av följande kod?

```
int sum = 1;
for (int i=1; i<3; i++);
sum +=1;
System.out.println(sum);
```

2	Svår! Ingen har får sänkt betyg pga denna fråga.

f) Hur många objekt skapas minst av koden

```
Turtle[] a = new Turtle[3];
ArrayList<Turtle> t = new ArrayList<Turtle>();
t.add(new Turtle(new World());
String s = "Ciao!";
```

5	Även strängar är objekt!

g) Givet koden

```
public void foo(int x) {
   int a = x*x;
   this.y = fie(a+2);
}
```

Ange alla

1)	formella parametrar	x
2)	aktuella parametrar	a+2
3)	lokala variabler	a
4)	instanstansvariabler	y

För att denna uppgift ska betraktas som godkänd bör du ha minst 5 helt rätta deluppgifter.

Vid tävlingar med individuella starter (t ex orientering och vissa skidlopp) vill man hålla reda på deltagarnas starttider (för att veta när de ska starta) samt deras mellantider och sluttider.
Tiderna räknas (för enkelhetens skull) i sekunder (heltal) och tävlingens klocka börjar med 0 när den första startar.
Klassen `Competitor` ska representera en deltagare med namn, starttid och en arraylista med mellantider och sluttid. Den som har lägst värde på sista platsen i arraylistan har alltså vunnit tävlingen.
Exempel: Om en tävlande har starttiden 60 och arraylistan `[103, 201, 306, 402, 505]` så passerade hen första kontrollen klockan 163, andra kontrollen klockan 261 och gick i mål klockan 565.

A2. Deklarera instansvariablerna `name` (en sträng), `startTime` (ett heltal) och en arraylista `time` med heltalsvärden för mellantider och tid i mål.

```
private String name;
private int startTime;
private ArrayList<Integer> time;
```

A3. Skriv färdigt nedanstående konstruktor som tar emot namn och starttid. Konstruktorn ska också skapa arraylistan.

```
public Competitor(String name, int startTime) {

    this.name = name;
    this.startTime = startTime;
    this.time = new ArrayList<Integer>();

}
```

A4. Skriv färdigt metoden `addTime(int clock)` som lägger in en ny mellantid (eller sluttid) sist i arraylistan. Inparametern `clock` är tävlingsklockans värde. Man måste alltså ta hänsyn till starttiden för att beräkna värdet som ska läggas in. Se den inledande diskussionen!

```
public void addTime(int clock) {

    time.add(clock - startTime);

}
```

A5. Skriv `toString`-metoden i klassen `Competitor`! Om inga mellantider är lagrade ska metoden bara returnera namn och starttid.

Exempel: `"Kalla 90"`.

Om det finns minst en lagrad mellantid ska även arraylistan med mellantider inkluderas i resultatet.

Exempel: `"Kalla 90 [123, 312, 431, 784]"`

```
public String toString() {
```

```
   String res = String.format("%10s %4d \t", name, startTime);
   if (time.size() > 0) {
      res += time;
   }
   return res;
```

```
}
```

A6. Skriv en annan `toString`-metod som tar emot en heltalsparameter i. Metoden ska returnera en sträng bestående av namnet och den tid den tävlande hade som i-*te* mellantid.

Exempel: Om `toString()` för personen returnerar `"Kalla 90 [123, 312, 431, 784]"` ska `toString(0)` returnera `"Kalla 123"` och `toString(1)` returnera `"Kalla 312"`.

```
public String toString(int i) {
```

```
   return  String.format("%10s %4d", name, time.get(i));
```

```
}
```

A7. Skriv en färdigt metoden `randomTime()` som skapar och returnerar ett slumpmässigt heltal i det slutna intervallet $[95, 105]$. Alla 11 värden ska ha lika stor sannolikhet.

```
public static int randomTime() {

    return (int)(95 + 11*Math.random());

}
```

A8. Skriv färdigt metoden `simulate` som fyller på tidlistan med med n slumpmässiga tider. Tiden för varje delsträcka ska beräknas med metoden `randomTime`.

Exempel: Anropet `simulate(5)` ska kunna ge en mellantidslista med innehållet $[98, 197, 302, 401, 502]$

```
public void simulate(int n) {

    time.add(0, randomTime());
    for (int i=1; i<n; i++) {
      int t =  randomTime() + time.get(i-1);
      time.add(t);
    }

}
```

De resterande uppgifterna på A-delen handlar om att lägga in kod i en `main`-metod som demonstrerar användningen av en del av ovanstående metoder. Metoden inleds så här:

```
public static void main(String[] args) {
   String[] names = {"Kalla", "Johaug", "Nilsson", "Flugstad"};
```

Den kod du skriver ska fungera även om antalet strängar och innehållet i dessa ändras.

Exempel på utskrift från den färdiga `main`-metoden (med svaret 5 på den inledande frågan):

```
Antal tider? 5
Kalla 0 [97, 200, 299, 397, 501]
Johaug 30 [96, 198, 296, 391, 487]
Nilsson 60 [103, 200, 303, 408, 509]
Flugstad 90 [97, 195, 292, 395, 497]
```

A9. *Deklarera* och *skapa* en *array* med namnet `skiers`. Arrayen ska lagra `Competitor`-objekt och kunna rymma lika många som det finns strängar i arrayen `names`.

```
Competitor[] skiers = new Competitor[names.length];
```

A10. Fyll arrayen med `Competitor`-objekt med namn enligt namnlistan ovan. De tävlande ska starta med 30 sekunders intervall.

```
for (int i = 0; i<names.length; i++) {
   skiers[i] = new Competitor(names[i], 30*i);
}
```

A11. Skapa ett `Scanner`-objekt för att läsa från tangentbordet. Skriv en fråga till användaren om hur många tider som ska läggas in och läs in svaret (heltal).

```
System.out.print("Antal tider? ");
Scanner scan = new Scanner(System.in);
int n = scan.nextInt();
```

A12. Iterera över den skapade arrayen. Använd metoden `simulate` och för att lägga till det antal mellantider som angavs som svar på ovanstående fråga.

Skriv också ut varje objekt.

```
for (Competitor s: skiers) {
   s.simulate(n);
   System.out.println(s);
}
```

Del B (för betyg 4 och 5)

Svaren skrivs på lösa papper med ny uppgift på nytt papper.

B1. Skriv en hjälpmetod `public double askDouble(String str)` för att läsa in ett flyttal från tangentbordet. Metoden ska skriva strängen i `str` som en ledtext. Om det som skrivs på tangentbordet inte går att tolka som ett flyttal (`double`) ska en felutskrift ges och användaren tillfrågas igen.

Exempel på kod och resulterande konversation. Programmets utskrifter i grönt, användarens inskrifter i rött.

```
double x = askDouble("Give the first: ");
System.out.println("First ok  : " + x);
double y = askDouble("Give the second: ");
System.out.println("Second ok : " + y);
```

```
Give the first: x
Not a double! Try again: x 21
Not a double! Try again: 25 7
First ok : 25.0
Give the second: zzz 47
Not a double! Try again: 12
Second ok : 12.0
```

```
public static double askAnd(String prompt) {
   System.out.print(prompt);
   Scanner s = new Scanner(System.in);
   while (!s.hasNextDouble()) {
      s.nextLine();
      System.out.print("Not a double! Try again: ");
   }
   return s.nextDouble();
}
```

B2. Skriv metoden `int[] twoSmallest(int[] a)` som skapar och returnerar en array med de två minsta värdena (i ordning med det minsta först) från arrayen `a`.

Exempel: Anropet `twoSmallest(new int[]{3, 5, 2, 8, 1})` ska returnera arrayen {`1, 2`}.

```
public static int[] twoSmallest(int[] a) {
   assert a.length >= 2;
   int[] res = new int[2];
   res[0] = Math.min(a[0], a[1]);
   res[1] = Math.max(a[0], a[1]);
   for (int i = 2; i < a.length; i++) {
      if (a[i] < res[1]) {
         res[0] = Math.min(res[0], a[i]);
         res[1] = Math.max(res[0], a[i]);
      }
   }
   return res;
}
```

B3. Skriv metoden `public ArrayList<String>readLines(String filename)` som läser rader från filen med namnet `filename`. Varje rad ska läggas in i en arraylist med strängar och denna arraylist ska returneras som värde.

```
public static ArrayList<String> readLines(String filename) throws IOException {
   FileReader reader = new FileReader(filename);
   Scanner scan = new Scanner(reader);
   ArrayList<String> names = new ArrayList<String>();
   while(scan.hasNextLine()) {
      names.add(scan.nextLine());
   }
   scan.close();
   reader.close();
   return names;
}
```

Klassen `Competition` ska användas för en tävling med ett antal deltagare. Klassen har två instansvariabler: en arraylista med deltagare (`Competitor`-objekt) och ett heltal som anger hur många kontrollstationer (tidtagningspunkter) som tävlingen har.
Klassen inklusive en `main`-metod med utskrifter finns på nästa sida.

B4. Skriv metoden `printCP(int i)` som listar ställningen vid den *i:te* stationen. Deltagarna ska listas i tidsordning med den snabbaste först. Exempel: Anropet `printCP(1)` ska kunna resultera i denna utskrift:

```
Checkpoint 1
Andersson  193
Karlsson   193
Flugstad   195
Jacobsen   198
Johaug     200
Kalla      200
Nilsson    201
Weng       201
```

```
   public void printCP(int ident) {
      System.out.println("\nOrder at checkpoint " + ident);
      ArrayList<Competitor> ordered = new ArrayList<Competitor>();
      for (Competitor s: competitors) {
         int index = 0;
         while (index < ordered.size() &&
                s.usedTime(ident) > ordered.get(index).usedTime(ident)) {
            index++;
         }
         ordered.add(index, s);
      }
      for (Competitor s : ordered) {
         System.out.println(s.toString(ident));
      }
   }
```

Samt, i klassen `competitor`:

```
   public int usedTime(int i) {
     return time.get(i);
   }
```

B5. Skriv konstruktorn `Competition(String filename, int interval, int checkpoints)`. Konstruktorn ska läsa in namn (ett per rad) från filen `filename`, skapa `Competitor`-objekt utifrån dessa namn samt lägga in dem i *slumpmässig* ordning i arraylistan. Den första i listan får starttid 0, nästa *interval*, nästa *2*interval* osv. Se körexemplen på nästa sida!

```
public Competition(String filename, int interval, int checkpoints)
    throws IOException
 {
    this.checkpoints = checkpoints;
    this.competitors = new ArrayList<Competitor>();

    ArrayList<String> names = readLines(filename);   // Metod från uppgift B3
    int startTime = 0;
    while (!names.isEmpty()) {
       int ind = (int)(Math.random()*names.size());
       String name = names.remove(ind);
       Competitor s = new Competitor(name, startTime);
       competitors.add(s);
       startTime += interval;
    }
 }
```

B6 Vissa av metoderna i uppgifterna B1 – B4 kan/bör deklareras `static`. Vilka? Vad betyder det?

En `static`-metod hör inte till individuella objekt utan är gemensam för klassen som helhet. Den kan således inte använda några instansvariabler (utan att gå via en objektreferens).

Metoderna i B1, B2 och B3 bör deklareras `static` eftersom de inte använder sig av några instansvariabler.

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.*;

public class Competition {
   private ArrayList<Competitor> competitors;
   private int checkpoints;

   public Competition(String filename, int interval, int checkpoints)
      throws IOException
   { ... }                                                  // Uppgift B5

   public void simulate() {
      for (Competitor s: competitors) {
         s.simulate(checkpoints);
      }
   }

   public void printAll() {
      System.out.println("\nAll results");
      for (Competitor s:competitors) {
         System.out.println(s);
      }
   }

   public void printCP(int ident) { ... }                // Uppgift B4

   public  static void main(String[] a) throws IOException {
      int checkpoints = 4;
      int interval = 30;
      Competition sr = new Competition("competitors.txt", interval, checkpoints);
      System.out.println("Start order:");
      for (Competitor s : sr.competitors) {
         System.out.println(s.toString());
      }
      sr.simulate();
      sr.printAll();
      sr.printCP(1);
      sr.printCP(2)
   }
}
```

När programmet körs kan det resultera i nedanstående utskrifter. Av utrymmestekniska skäl har de olika utskrifterna placerats bredvid varandra i stället för efter varandra som det blir när programmet verkligen körs.

```
Start order:
Weng          0
Nilsson      30
Karlsson     60
Andersson    90
Kalla       120
Jacobsen    150
Flugstad    180
Johaug      210
```

```
All results
Weng          0 [101, 201, 297]
Nilsson      30 [101, 201, 305]
Karlsson     60 [98, 193, 291]
Andersson    90 [98, 193, 296]
Kalla       120 [95, 200, 298]
Jacobsen    150 [96, 198, 302]
Flugstad    180 [98, 195, 291]
Johaug      210 [98, 200, 299]
```

```
Checkpoint 1
Andersson   193
Karlsson    193
Flugstad    195
Jacobsen    198
Johaug      200
Kalla       200
Nilsson     201
Weng        201
```

```
Checkpoint 2
Flugstad    291
Karlsson    291
Andersson   296
Weng        297
Kalla       298
Johaug      299
Jacobsen    302
Nilsson     305
```